Reconciling White-Box and Black-Box Perspectives on Behavioral Self-adaptation

Bruni, Roberto; Corradini, Andrea; Gadducci, Fabio; Hölzl, Matthias; Lluch Lafuente, Alberto; Vandin, Andrea; Wirsing, Martin

Published in:
Software Engineering for Collective Autonomic Systems

Link to article, DOI:
10.1007/978-3-319-16310-9_4

Publication date:
2015

Document Version
Peer reviewed version

Reconciling White-Box and Black-Box Perspectives on Behavioural Self-Adaptation*

Roberto Bruni[1], Andrea Corradini[1], Fabio Gadducci[1], Matthias Hölzl[2], Alberto Lluch Lafuente[3], Andrea Vandin[4], and Martin Wirsing[2]

[1] Department of Computer Science, University of Pisa, Italy
[2] LMU, Ludwig Maximilians University of Munich, Germany
[3] DTU Compute, Technical University of Denmark, Denmark
[4] Electronics and Computer Science, University of Southampton, UK

Abstract. This paper propose to reconcile two perspectives on behavioural adaptation commonly taken at different stages of the engineering of autonomic computing systems. Requirements engineering activities often take a *black-box* perspective: A system is considered to be adaptive with respect to an environment whenever the system is able to satisfy its goals irrespectively of the environment disturbations. Modelling and programming engineering activities often take a *white-box* perspective: A system is equipped with suitable adaptation mechanisms and its behaviour is classified as adaptive depending on whether the adaptation mechanisms are enacted or not. The proposed approach reconciles black- and white-box perspectives by proposing several notions of coherence between the adaptivity as observed by the two perspectives: These notions provide useful criteria for the system developer to assess and possibly modify the adaptation requirements, models and programs of an autonomic system.

Keywords: Autonomic Computing, Behavioural Adaptation, Requirements Engineering, Software Engineering, Linear-time Properties, Games

1 Introduction

Autonomic systems operating in highly variable, even unpredictable, environments must be *self-adaptive*. Unfortunately, there is not a widely accepted agreement on a foundational model for adaptivity. Already in the early sixties Lofti Zadeh [27] claimed that *"it is very difficult – perhaps impossible – to find a way of characterizing in concrete terms the large variety of ways in which adaptive behavior can be realized"* and was pessimistic regarding the possibility to obtain a unifying definition due to the inherent difficulty of subsuming under the same hood both the *external* manifestations of adaptive systems (*black-box* adaptivity) and the *internal* mechanisms by which adaptation is achieved (*white-box* adaptivity).

* This research was supported by the European project IP 257414 (ASCENS) and by the Italian project PRIN 2010LHT4KM (CINA).

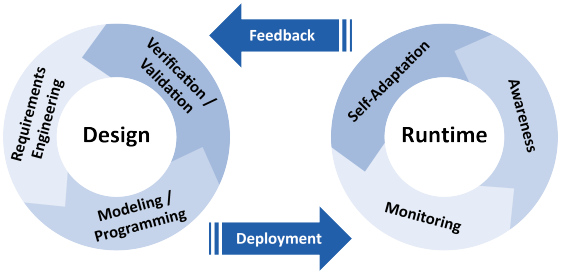


Fig. 1. The ASCENS life cycle for autonomic software component ensembles

These two perspectives persist today. Often, requirements engineering activities take the black-box one, by expressing requirements as goals to be achieved in some environments (cf. Chapter III.3 [26]). The system is adaptive to an environment if it may satisfy the goal in that environment. Sometimes, pairs of environments and goals is considered, if it makes sense to consider different goals in different environments. An example of these approaches is in the ASCENS [4] methodology (cf. Chapter III.1 [13]) to the engineering of autonomic systems (cf. Fig. 1): The *General Ensemble Model* (GEM) [14], a formalisation of the SOTA [1] approach to requirements engineering, takes a black-box perspective.

Later design activities such as modelling and programming tend to take the white-box perspective by focusing on the realisation of adaptation mechanisms, often based on linguistic or architectural techniques. A widely used approach is to clearly separate the functional and adaptation logics when structuring the behaviour of a system. Thus, computations are classified according to the presence of normal steps aimed at realising the functional logic and adaptation steps aimed at adapting the system's behaviour. An archetypal approach are Adaptable Interface Automata (AIAs) [6], a formalisation of white-box perspectives for autonomic systems like CODA [7] developed within ASCENS [4].

Contribution. Clearly, adaptation remains a subjective concept in both perspectives. The fundamental difference lies in who is responsible of declaring whether a system is adaptive or not: the requirements engineer (black-box) or the system engineer (white-box). Ideally, both perspectives should be coherent. This paper proposes an approach to reconcile black- and white-box approaches to behavioural adaptation in autonomic systems and to asses their coherence.

The key idea is that the white-box perspective allows to classify and quantify the behaviours of the system in an environment according to the presence of actual adaptations. Mismatches between black- and white-box perspectives may arise, e.g. if the system satisfies its goals in an environment, but no adaptation is observed. Mismatches can be used to improve the system, e.g. by disabling computationally expensive adaptation mechanisms based on monitoring and awareness (cf. Fig. 1). Moreover, mismatches may provide information to requirements and system engineers, since they can help to asses and modify specifications.

Our presentation focuses on the above mentioned approaches developed within the ASCENS project [4], respectively GEM and AIAs. However, the methodology we introduce is general enough to be applicable in a wide range of approaches to the engineering of autonomic systems.

Structure of the Paper. Section 2 briefly presents a case study from the ASCENS project that we use for illustrative purposes throughout the paper. Section 3 recalls two archetypal approaches to adaptation based on black-box and white-box perspectives, namely GEM (Section 3.1) and AIAs (Section 3.2). Section 4 presents our approach to reconcile them. Section 5 discusses related works. Section 6 concludes the paper and outlines opportunities for future research.

2 A Robot Rescue Case Study

We shall use the *robot rescue scenario* (cf. Chapter IV.2 [21]) from the ASCENS case studies (cf. Chapter IV.1 [25]) as a running example throughout the paper. Fig. 2 illustrates this scenario, where a swarm of rescuer robots (represented as black circles with a grip) has to rescue victims (represented as stars) of a natural or industrial disaster (toxic waves in the figure), possibly cooperating to secure the area first (building a protection barrier in the figure) and then pulling the victim with the grippers until a safe place (e.g. Home in the figure) is reached.

To keep the scenario as simple as possible we limit us in some cases to a single rescuer robot and a single victim that has to be rescued. We assume that the locations of the victim and the robot are expressed as Cartesian coordinates: The robot can navigate the environment with constraints specified by kinematic equations and pick up objects in the environment according to certain physical constraints. The main requirement of the system is that the robot transports the victim to a rescue zone, but often a purely goal-based description is not sufficient to capture the real requirements.

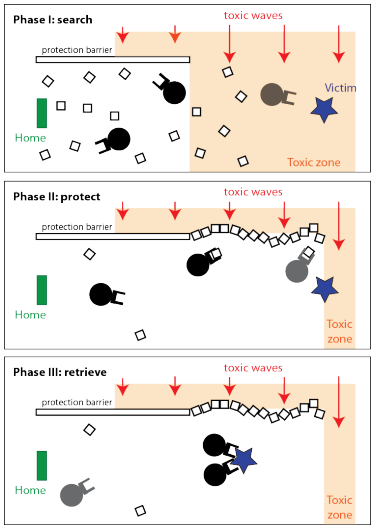


Fig. 2. Robot Rescue Scenario

We assume that one of the main question of interest is to determine whether the rescuer robot is capable of rescuing the victim under certain conditions on the environment. This is, as we shall see in Section 3.1, precisely a question about the adaptivity properties from the black-box perspective. For instance, if the victim has a weight of 80 Kg and the robot is capable of lifting only 20 Kg, then it is simply not possible for the robot to complete the rescue mission, given reasonable assumptions about the operating environment. However, an autonomic robot with these specifications operating on the moon would be able to lift the victim, and likewise a subaqueous robot.

The interplay between system and environment is not only important for the general structure of the system, it remains relevant as designers progress to more and more detailed system designs and their inherent trade-offs: A biped robot that is in principle capable of lifting a victim might not be able to do so in a marsh whereas a tracked vehicle might be able to. On the other hand, a tracked vehicle might not be capable of navigating stairs that pose no problem for the biped. A quadcopter might be able to lift a victim if the location is at sea level, but not if it is high in the mountains. Of course, given a specification of the problem the system designers and programmers have to decide how to actually solve the problem from an architectural and algorithmic point of view. This very same scenario has been tackled in the ASCENS project with a variety of techniques, from aspect-oriented programming to high-level policies, to meta-programming techniques (see e.g. the long discussion in [10] and the references therein).

3 Black-Box and White-Box Adaptation

This section offers a gentle introduction to two archetypal examples of black-box and white-box approaches to adaptation in autonomic computing systems, namely GEM (Section 3.1) and AIAs (Section 3.2). The presentation of both approaches has been simplified to focus on their essential features.

3.1 A Black-Box Perspective on Adaptive Systems

GEM [14], the *General Ensemble Model*, is a formalisation of the statespace-based SOTA approach to requirements engineering [1]. GEM models describe the possible behaviours of a system as trajectories through its phase space[5] while the system is interacting with an environment, i.e., a system consists of a preordered *time structure* T, a phase space X and a set of trajectories in $T \to X$.[6] T might be as simple as the natural numbers with the usual order, while in our example the state space might be the position of each robot. Since they are used during the analysis or the high-level design of a system, the initial GEM models of a system typically have very little information about its internal structure. Chapter III.3 [26] in this volume discusses the Ensemble Development Life Cycle (EDLC) and the role that SOTA plays in the overall development process.

We illustrate the kind of GEM models that might be built during the development of a system like our scenario in Section 2, with some examples: The developer might start the initial system inception by defining simple and abstract requirements in the form of GEM models that are meant to analyse whether the proposed system is feasible at all. These models can be either discrete-state or continuous. For instance, for large ensembles we might approximate the number r of rescue robots that are still properly working after time t by the differential

[5] In this section we use the terms phase space and state space interchangeably.

[6] GEM actually allows the definition of more general types of systems, and ensembles in GEM are defined in a manner that is isomorphic but not identical to the one given here. The definition given above corresponds to *time ensembles*.

equation $\dot{r}(t) = -\alpha r(t) + c$, where α is the rate at which robots can be damaged (e.g. by toxic waves), c is a constant rate of newly produced robots, and $\dot{r}$ is the derivation dr/dt. After the feasibility of the system has been determined, a high-level GEM model for the rescue scenario might describe the state of the system in an abstract manner, e.g. by specifying the aforementioned Cartesian locations and movement constraints.

In general the phase space X becomes unwieldy as the systems become larger, therefore GEM allows developers to define individual components of a system on their own, and to combine them using *combination operators* that construct a new state space from the state spaces of the components. Those operators are denoted by $\otimes$. In the example, the system's designers might define two components, the robot S_r and the victim S_v, to investigate whether a robot can transport victims under the given environment conditions. To be useful the state of these components has to be sufficiently complete to allow a full description of the relevant aspects of the system behaviour. The state X_r of the robot would probably include state variables for the robot's velocity and orientation, its position, the configuration of its gripper, its carrying capacity and so on. Similarly, the state of a victim would contain information about its physical properties, e.g., weight and how the victim's body behaves when it is picked up by the robot. By applying a suitable combination operator, designers can build a component $S_r \otimes S_v$ that represents a robot carrying a victim. In the combined system, state variables of the individual components that are no longer necessary to describe the combined system can be suppressed and new state variables that result from the combination can be introduced. For example, the configuration of the robot's gripper and the physical properties of the victim's body may be replaced in $S_r \otimes S_v$ by state parameters that specify the manoeuvrability of the robot while it is carrying the victim.

In a simple version of the rescue scenario, the main requirement for each rescuer robot can be described in a purely goal-based manner: Transport the victim to a safe zone. On the other hand, such a purely goal-based description is often not sufficient to capture the real requirements. Therefore, GEM allows developers to describe the quality of a solution (called its *utility* in GEM) using real numbers. For example, the utility of the robot might be defined as the time it needs to rescue the victim, possibly taking into account factors such as the exposure of the victim to environmental hazards during the transport phase.

The GEM *Approach to Black-Box Adaptation.* GEM defines a notion of adaptation, which we call *black-box adaptation*, that is applicable to models in the early stages of requirements engineering of autonomic computing systems (cf. the leftmost cycle in Fig 1). It is clear that at those stages the focus of the definition has to rely on the "circumstances" aspect of adaptation and not on the "change of the system", since little or no information about the system's internal structure is usually available at this stage of the development process. Nevertheless there are some interesting adaptation-related questions that can be answered by looking at the intrinsic capabilities of a *system* S and its *environment* η.

As we already advanced in the previous section, the most obvious question is perhaps "Is the system in principle capable of satisfying the requirements?". More precisely, it is mandatory to specify the range of environments $\mathcal{E}$ under consideration and ask for which environments in $\mathcal{E}$ a given system can fulfil the requirements. This is the basic idea underlying black-box adaptation.

While focusing on systems and environments would be sufficient from a theoretical point of view, models are often not detailed enough to contain all the information required to analyse the behaviour of the combination. For example, the ability to recognise robots might be reduced because of lighting conditions in the environment; however the robot's model might not contain sufficient detail to represent this when combining it with an environment. GEM therefore introduces a third component into the ensemble model that describes the interaction of the system with the environment. In GEM this component is called the *network* ν, although in the rescue example "system/environment fit" might better describe the role of ν. For example, the performance of a robot's camera obviously depends on the lighting conditions, but in many cases the modelers do not want to specify the system in enough detail to compute the effect of lighting on the camera. To model the decreased camera quality under the expected lighting condition, the camera can instead be combined with the environment using a noisy or lossy communication channel in ν, without adjusting the models of the system or the environment themselves.

The GEM *Formalisation of Black-Box Adaptation.* We can now describe black-box adaptation in slightly more detail.

Definition 1 (adaptation domain and requirements). *Given a set of environments $\mathcal{E}$, networks $\mathcal{N}$ and goals $\mathcal{G}$, an* adaptation domain *$\mathcal{A}$ is a subset of $\mathcal{E} \times \mathcal{N} \times \mathcal{G}$, i.e., a set of triples (η, ν, γ) called an* adaptation requirements.

An adaptation domain is therefore a package consisting of environments, networks and goals that are of interest to the system designer. Adaptation domains provide a basic signature for adaptation requirements that can be instantiated in several ways as we shall see further in this paper. A concrete example of this is the requirements specification language SOTA [1].

Concrete GEM models where adaptation domains are given a precise semantics can be subject to different kinds of analysis. A typical example is requirements satisfiability. For instance, the adaptation domain $\mathcal{A} = \{(\eta, \nu, \gamma), (\eta, \nu, \neg\gamma)\}$ is unlikely to be satisfiable under any reasonable semantics for $\neg$.

Of course, given a satisfiable adaptation domain, the question remains whether a given system can adapt to it. This concept is formalised as follows.

Definition 2 (adaptation to a domain). *Let $S \otimes \eta \otimes \nu \models \gamma$ denote that a system S* fulfils *a goal γ when in environment η with network ν. We say a system S* can adapt to *an adaptation domain $\mathcal{A}$, written $S \Vdash \mathcal{A}$, iff $\forall(\eta, \nu, \gamma) \in \mathcal{A} \,.\, S \otimes \eta \otimes \nu \models \gamma$.*

Again some of the concepts such as goal fulfillment $\models$ and composition $\otimes$ are left underspecified on purpose. They can be instantiated in many different ways. For instance, in this paper we shall see a trace-based semantics for $\models$ and automata synchronisation semantics for $\otimes$, which turn out to be very useful for reconciling black- and white-box perspectives.

Very often, one needs to compare the adaptation ability of different systems, which may correspond to different early designs of the same system. Recall, for instance, our case study (cf. Section 2) and the example of biped and tracked robots, where systems may make different trade-offs and therefore be able to deal with different adaptation requirements or domains. For instance, given a fixed adaptation domain $\mathcal{A}$, a system S_1 may be able to satisfy *every* adaptation requirement system in $\mathcal{A}$ that S_2 can satisfy; in that case we say S_1 is *at least as adaptive to* $\mathcal{A}$ as S_2. But it can be also the case that a system S_1 may be able to operate in *every* adaptation domain of interest in which system S_2 can operate; in that case we say S_1 is *at least as adaptive* as S_2. If S_1 can additionally adapt to at least one adaptation domain to which S_2 cannot adapt we call S_1 *more adaptive* than S_2. More formally, we define an adaptation space $\mathfrak{A}$ as a family of adaptation domains, $\mathfrak{A} \subseteq \mathfrak{P}(\mathcal{E} \times \mathcal{N} \times \mathcal{G})$. For any adaptation space we define a preorder of adaptivity for systems as follows.

Definition 3 (adaptation pre-orders). *Let $\mathfrak{A}$ be an adaptation space and let $\mathcal{A} \in \mathfrak{A}$ be one of its adaptation domains. The $\mathcal{A}$-adaptation pre-order $\sqsubseteq_{\mathcal{A}} \in S \times S$ is the relation among systems defined by*

$$S \sqsubseteq_{\mathcal{A}} S' \iff \forall(\eta, \nu, \gamma) \in \mathcal{A} \,.\, S \otimes \eta \otimes \nu \models \gamma \implies S' \otimes \eta \otimes \nu \models \gamma$$

The adaptation pre-order $\sqsubseteq_{\mathfrak{A}} \in S \times S$ is the relation among systems defined by

$$S \sqsubseteq_{\mathfrak{A}} S' \iff \forall \mathcal{A} \in \mathfrak{A} : S \Vdash \mathcal{A} \implies S' \Vdash \mathcal{A}$$

It is worth to remark that $S \sqsubseteq_{\mathfrak{A}} S'$ implies $\forall \mathcal{A} \in \mathfrak{A} \,.\, S \sqsubseteq_{\mathcal{A}} S'$ but the contrary is not true. A trivial example is one in which the adaptation space $\mathfrak{A}$ is just $\{\mathcal{A}\}$, neither S nor S' adapt to $\mathcal{A}$ but $S \sqsubseteq_{\mathcal{A}} S'$, i.e. S' can satisfy more requirements than S but they both adapt to the same set of domains, namely none.

It is easy to extend those relations to the case of utility-based systems; intuitively a system S' is more adaptive than a system S given an adaptation domain $\mathcal{A}$ if the utility of S' is at least as high as that of S for every element of $\mathcal{A}$, and higher for at least one element of $\mathcal{A}$. This definition is extended to adaptation spaces in the obvious way.

All in all, GEM's approach to adaptation requirements engineering from a black-box perspective provides a simple yet useful tool for designers to analyse various aspects of early system specifications. For instance, GEM models can be used to compare various early systems designs in terms of their adaptation abilities to cope with the desired requirements; adaptation spaces can also be used to check whether it is at all possible to satisfy all adaptation requirements, or whether certain requirements are expensive to implement.

3.2 A White-Box Perspective on Adaptive Systems

White-box perspectives on adaptation shift the attention to later stages of system design (e.g. Modelling and Programming in Fig. 1) and allow one to specify or inspect (part of) the internal structure of a system in order to offer a clear *separation of concerns* to distinguish changes of behaviour that are part of the application or functional logic from those which realise the adaptation logic. This section summarises the CoDa approach to adaptation and one of its formalisations, namely *Adaptable Transition Systems* [6].

The CoDa *Approach to White-Box Adaptation.* In general, the behaviour of a system is governed by a program and according to the traditional, basic view, a program is made of *control* (i.e. algorithms) and *data*. The conceptual notion of adaptation proposed in [7] requires to identify *control data* which can be changed to *adapt* the system behaviour. *Adaptation* is, hence, the run-time modification of such control data. According to this notion, a system is *adaptable* if it has a distinguished collection of control data that can be modified at run-time, *adaptive* if it is adaptable and its control data are modified at run-time, at least in some of its executions, and *self-adaptive* if it modifies its own control data at run-time. So, essentially, the CoDa approach to measure the adaptation ability of a system is to identify control data and observe its modification in the system's behaviours. For example, the adaptation mechanisms of robots like the ones in our scenario are sometimes based on the use of operation modes. Each mode of operation is tailored to some specific class of situation and a high-level controller adapts the behaviour of the robot by switching between modes of operation. In this case, the control data is precisely the data defining the current mode of operation.

Several programming paradigms and reference models have been proposed for adaptive systems. A notable example is the Context Oriented Programming paradigm, where the contexts of execution and code variations are first-class citizens that can be used to structure the adaptation logic in a disciplined way [24]. This paradigm has also influenced the many programming and modelling approaches developed within the ASCENS project [4] among which we cite the *Service Component Ensemble Language* (see [11] and also Chapter I.1 [19]) and the architectural approach of [9]. Nevertheless, it is not the choice of the programming language what makes a program adaptive: any computational model or programming language can be used to implement an adaptive system, just by identifying the part of the data that governs the adaptation logic, that is the control data. Consequently, the nature of control data can vary considerably, including all possible ways of encapsulating behaviour: from simple configuration parameters to a complete representation of the program in execution that can be modified at run-time, as it is typical of computational models that support meta-programming or reflective features.

Adaptable Transition Systems: A Foundational Model for CoDa. Adaptable Interface Automata (AIAs) are a model of adaptive component-based systems built upon *interface automata* [2,3]. AIAs are an incarnation of the more general

concept of adaptable transition systems. The key feature of AIAs are *control propositions* evaluated on states, the formal counterpart of control data. The choice of such propositions is arbitrary but it imposes a clear separation between ordinary behaviours and adaptive ones.

Interface automata were introduced in [3] as a framework for component-based design and verification. Following [2], an interface automaton P is a tuple $\langle V, V^i, \mathcal{A}^I, \mathcal{A}^O, \mathcal{T} \rangle$, where V is a set of states; $V^i \subseteq V$ is the set of initial states, which contains at most one element (if V^i is empty then P is called *empty*); $\mathcal{A}^I$ and $\mathcal{A}^O$ are two disjoint sets of *input* and *output actions* (we denote by $\mathcal{A} = \mathcal{A}^I \cup \mathcal{A}^O$ the set of all actions); and $\mathcal{T} \subseteq V \times \mathcal{A} \times V$ is a deterministic set of *transitions* (also called *steps*), i.e. for any two transitions $(u, a, v) \in \mathcal{T}$ and $(u, a, v') \in \mathcal{T}$ we have that $v = v'$. An interface automaton can represent, for instance, the behavioural model of one of our robots, where actions represent interactions with the environment and the distinction between input and output reflects the flow of information between them.

As usual, a transition (u, a, v) can be denoted by $u \xrightarrow{a} v$. The absence of non-determinism is not essential for the purposes of this paper, but it plays a fundamental role for the feasibility of control synthesis. Given $\mathcal{B} \subseteq \mathcal{A}$, we sometimes use $P_{|_\mathcal{B}}$ to denote the automaton obtained by restricting the set of steps to those whose action is in $\mathcal{B}$. Also, the set of actions in $\mathcal{B}$ labelling the outgoing transitions of a state u is denoted by $\mathcal{B}(u)$. A *computation* ρ of an interface automaton P is a finite or infinite sequence of consecutive transitions $\{u_i \xrightarrow{a_i} u_{i+1})\}_{i<n}$ from $\mathcal{T}$ (thus n can be ω).

In our case study, an interface automaton could represent the overall behaviour of a robot, possibly obtained as the result of composing different components such as controllers and sensors. The main idea of AIAs is to exhibit the minimal amount of information that allows one to distinguish the ordinary behaviours aimed at realising the application logic of the system (e.g. a robot) from the behaviours aimed at realising the system's adaptation. AIAs extend interface automata with atomic propositions (state observations), some of which are called *control propositions* and play the role of the control data of [7].

Definition 4 (adaptable interface automata). *An* adaptable interface automaton *(*AIA*) is a tuple* $\langle P, \Phi, l, \Phi^c \rangle$ *such that* $P = \langle V, V^i, \mathcal{A}^I, \mathcal{A}^O, \mathcal{T} \rangle$ *is an interface automaton;* Φ *is a set of atomic propositions,* $l : V \to 2^\Phi$ *is a labelling function mapping states to sets of propositions; and* $\Phi^c \subseteq \Phi$ *is a distinguished subset of* control propositions*.*

We call P an AIA with underlying interface automaton P, whenever this introduces no ambiguity. Most of the ingredients of Interface Automata are trivially lifted to AIAs. This includes the notion of computation. Furthermore, we shall consider as well the usual notion of trace, denoting with $\mathit{Traces}(P)$ the set of traces of an AIA P, i.e. the projection of all computations of P on the state observation Φ. More precisely, if $u \xrightarrow{a} v$ is a transition of a computation, its projection on a trace will be the trace transition $l(u) \xrightarrow{a} l(v)$. Additional lifted concepts are those of compatibility and composition among AIAs. Indeed,

AIAs come equipped with a suitable composition operator, denoted $\otimes$, which essentially corresponds to lifting the composition operator on the underlying Interface Automata structure. We refer the interested reader to [6]. The choice of symbol $\otimes$ is not an accident: interpreting composition in GEM models as composition of AIAs is indeed part of our approach to reconcile both views on adaptation, as we shall explain later in Section 4.

The AIA *Approach To White-Box Adaptation.* A transition $u \xrightarrow{a} u' \in T$ is called an *adaptation* if it changes the control data, i.e. if there exists a proposition $\phi \in \Phi^c$ such that either $\phi \in l(u)$ and $\phi \notin l(u')$, or vice versa. Otherwise, it is called a *basic* transition. Consider for instance an AIA modelling one of the robots of our case study. Let us assume that its control propositions allow one to observe the robot's mode of operation, whose change is the way of implementing adaptive behaviours. Then the adaptive transitions will exactly correspond to those adaptive behaviours, i.e. switching between modes of operation. The basic behaviours would be the ordinary behaviour within a mode of operation. Clearly, the concept trivially lifts to trace transitions. An action $a \in A$ is called a *control action* if it labels at least one adaptation. The set of all control actions of an AIA P is denoted by $\mathcal{A}_P^C$.

Computations are classified according to the presence of adaptation steps.

Definition 5 (adaptive computations). *Let P be an* AIA *and ρ be a computation in P. We say that ρ is* basic *if it contains no adaptive transition, and it is* adaptive *otherwise.*

We will also use the concepts of *basic computation* starting at a state u and of *adaptation phase*, i.e. a maximal computation made of adaptive steps only. Again, this concept can be trivially lifted to traces.

The concept of adaptive trace can be formalised in temporal logics such as LTL as follows, $\psi_{adaptive} \equiv \neg \bigwedge_{\phi \in \Phi^c} (\phi \leftrightarrow \mathbf{G}\phi)$, i.e. it is not the case that all observable control data remain the same forever. This clearly enables the use of standard model checking techniques to analyse properties of adaptive systems (see e.g. the references and the discussion in [6]).

It is worth to remark that what distinguishes adaptive computations and adaptation phases are not the actions, because control actions may also label transitions that are not adaptations. However, very often an AIA has *coherent control*, meaning that the choice of control propositions is coherent with the induced set of control actions, in the sense that all the transitions labelled with control actions are adaptations. In the rest of the paper we assume that every system under consideration has coherent control.

The relationship between the set of control actions $\mathcal{A}_P^C$ and the alphabets $\mathcal{A}_P^I$ and $\mathcal{A}_P^O$ is arbitrary in general, but it could satisfy some pretty obvious constraints for specific classes of systems. Let P be an AIA. We say that P is *adaptable* if $\mathcal{A}_P^C \neq \emptyset$; *controllable* if $\mathcal{A}_P^C \cap \mathcal{A}_P^I \neq \emptyset$; *self-adaptive* if $\mathcal{A}_P^C \cap \mathcal{A}_P^O \neq \emptyset$. Intuitively, an AIA is *adaptable* if it has at least one control action, which means

that at least one transition is an adaptation. An adaptable AIA is *controllable* if control actions include some input actions and *self-adaptive* if control actions include some output actions (which are under the control of the AIA). From these notions we can derive others. For instance, we can say that an adaptable AIA is *fully self-adaptive* if $\mathcal{A}_P^C \cap \mathcal{A}_P^I = \emptyset$ (the AIA has full control over adaptations). Note that hybrid situations are possible as well, when control actions include both input actions (i.e. actions in $\mathcal{A}_P^I$) and output actions (i.e. actions in $\mathcal{A}_P^O$). In this case we have that P is both *self-adaptive* and *controllable*. In our case study we could expect to have fully self-adaptive ground rescue robots or partially adaptive ones that can be influenced from supervisor quadcopter robots.

Those notions can be lifted to computations in the obvious way.

Definition 6 (adapted computations). *Let P be an* AIA *and ρ be a computation in P. We say that ρ is* controlled *if it contains a transition which is an adaptation and corresponds to an input action, and it is* fully self-adapted *if all its adaptation transitions correspond to output actions.*

Such concepts may be formalised as well with modal logics. Of course, LTL cannot be directly applied since it is a pure state-based logic and here we need to observe actions, but one may use logics for double-labelled transition systems or standard encodings between action-labeled transitions systems (e.g. automata) and state-labelled transitions systems (e.g. Kripke structures).

4 Reconciling Black-Box and White-Box Adaptation

We present here our approach to reconcile white-box and black-box perspectives on adaptation. We assume that the black-box perspective on the systems under study is formalised by Gem models whose systems, environments and networks are suitably represented by AIAs, our reference model for white-box perspective. One of the key observations regarding the different treatments of adaptation on black-box and white-box perspectives is that adaptation in the former case is "asymmetric" in that it considers various environments in which a system may operate, but the system is defined by all its possible behaviours, irrespectively of whether they require internal changes or not. Tackling this asymmetry is one of the issues in the conciliation. We propose here two ways of reconciling white-box and black-box perspectives on adaptation. A first, simple one is based on a trace-based semantics of Gem and AIAs (cf. Section 4.1). The second one (cf. Section 4.2) is based on game semantics.

4.1 Trace-based Reconciliation

A key assumption to combine Gem and AIA concepts is to assume that Gem notion of property satisfaction is based on semantic relations on computations. For the sake of illustration, we consider the well-studied and popular notion of property satisfaction based on trace inclusion.

Definition 7 (trace-based adaptation). *Let S be a system and (η, ν, γ) be a requirement. We instantiate the fulfilment by S of a goal γ in an environment η with network ν as*

$$S \otimes \eta \otimes \nu \models \gamma \quad \textit{iff} \quad \mathit{Traces}(S \otimes \eta \otimes \nu) \subseteq \mathit{Traces}(\gamma)$$

Recall that trace inclusion amounts to the satisfaction relation of model checking linear-time temporal logics. This means that if goals are expressed as LTL properties (as SOTA requirements essentially are), checking the fulfilment of requirements can be done with efficient state-of-the art model checking techniques.

A consequence of our choice of trace based semantics is that we are instantiating the adaptation of a system S to an adaptation domain $\mathcal{A}$ as $S \Vdash \mathcal{A}$ iff $\forall(\eta, \nu, \gamma) \in \mathcal{A}. \mathit{Traces}(S \otimes \eta \otimes \nu) \subseteq \mathit{Traces}(\gamma)$. This trace-based notion is not only interesting from the theoretical point, but it allows us to re-use all the useful machinery of linear-time properties, including model checking. For instance, we may assume goals γ to be suitably specified in languages with linear-time semantics such as ω-regular expressions, linear-time temporal logic, and various forms of automata or transition systems. Moreover, the SOTA approach to requirements engineering for autonomic systems, for which GEM provides a formalisation, is indeed based on trace semantics and inclusion relations.

More interestingly, the trace-based semantics offers a common ground to compare and reason about properties of systems and their computations from both the black-box and white-box perspectives. The first interesting question to reconcile the two perspectives on a system is the following. Let S be a system that is adaptive to an adaptation domain $\mathcal{A}$ (i.e. $S \Vdash \mathcal{A}$). Are the behaviours of S that witness adaptation to $\mathcal{A}$ adaptive from the white-box point of view? This leads us to the concept of *coherent adaptation*.

Definition 8 (coherent adaptation). *Let S be a system and $\mathcal{A}$ be an adaptation domain. We say that S is* coherently adaptive *to $\mathcal{A}$ if $S \Vdash \mathcal{A}$ implies that for all $(\eta, \nu, \gamma) \in \mathcal{A}$ there is at least one trace $\rho \in \mathit{Traces}(S \otimes \eta \otimes \nu)$ that is adaptive.*

It is worth to remark once more that coherence can be boiled down to a standard model checking problem. Note that if a system S is not coherently adaptive to an adaptation domain $\mathcal{A}$ then the system is able to satisfy one goal in $\mathcal{A}$ without performing any transition observationally identified as adaptation (from the white-box perspective). This implies a certain mismatch between what is considered to be a normal operation environment from the system design point of view (i.e. the system S as an AIA) and from the requirements point of view (i.e. the adaptation domain $\mathcal{A}$). The system designer can then exploit this information for improving the design of the system. For instance, in the case of our case study, a lack of coherent adaptation could help the designer decide that the components of the robot software realising its adaptive behaviour are not necessary so that robots can be deployed with a light-weight version of the software, possibly consuming less computational resources and battery.

Further distinctions can be done by considering the presence/absence of adaptation transitions controlled by the environment: if all adaptation transitions correspond to output control actions of S, the system is *fully self-adaptive* to $\mathcal{A}$, even if it may not necessarily be *fully self-adaptive* in general.

Ideally, one would like to have a system S that is *coherently adaptive* and *fully self-adaptive* with respect to the adaptation domain $\mathcal{A}$ under consideration. That is, a system that has full control over its adaptation actions when operating in $\mathcal{A}$ and whose adaptation mechanisms are actually enacted in all the situations identified in the adaptation domain $\mathcal{A}$. When this is not the case the system designer can take the necessary actions as we exemplified above.

4.2 A Game-Based Reconciliation

The last section introduced the notion of coherent adaptation for systems represented by AIAs. This definition can be generalised to arbitrary SOTA/GEM models if we define control propositions Φ^c (see Section 3.2) as propositions in a suitable logic over the state space X. These propositions define a subset $X^c \subseteq X$ that we call the *control space* of the system. The notion of adaptive computation from Definition 5 can then be applied to segments of SOTA/GEM traces: Let T be the time domain of the model, $\theta : T \to X$ a trajectory in X, and $t_1, t_2 \in T$. We say that θ *is adaptive* in the interval $[t_1, t_2]$ if it intersects the control space in that interval, i.e., $\{\theta(t) \mid t \in [t_1, t_2]\} \cap X^c \neq \emptyset$; it is *purely adaptive* if it is contained in X^c, i.e., $\{\theta(t) \mid t \in [t_1, t_2]\} \subseteq X^c$; and it is *basic* if it is not adaptive. Definition 8 can be applied to arbitrary SOTA/GEM models in a similar manner.

An environment that reacts to the system behaviours but does not pursue its own goals is sometimes called (purely) *parametric* or *stochastic*; an environment in which other agents pursue their own goals (and thus may actively work to inhabit the ensemble goals) is called *game-theoretic* or *strategic*. Many interesting and practically occurring environments are strategic; in these environments it is useful to regard black-box adaptation as a game between two players *System* and *Environment*, moderated by a *Network* that determines properties such as the information available to each player during the game.

The structure of this game (e.g., whether the players move simultaneously or in an alternating manner) and the features of the combined state space X are given by a combination operator between the models of the *System* S, the *Environment* η and the *Network* ν. The moves of *System* and *Environment* are given by the models S and η, and their effect is described by the change to the combined state space X. The information about the moves of the other player is provided by ν (which may also contribute to the shared state space). Roughly speaking, S and η determine the moves in the game, whereas ν describes the information sets and thus, together with the combination operator, determines whether the game is one with concurrent or sequential moves and whether it is one of perfect or imperfect information. We call this way of specifying a GEM model M a *game-based presentation* (of M). In a game-based presentation we may say "M_1 wins against more opponents (from an adaptation domain $\mathcal{A}$) than M_2" instead of "M_1 is more adaptive than M_2."

SOTA/GEM themselves impose no specific organisation on models, and many SOTA/GEM models have a continuous time structure and large state spaces; therefore writing a game-based presentation of a model does not imply that the model becomes amenable to game-theoretic analysis. However, typical models are often structured in a particular way: The model is represented as a hierarchy of components, and components interact mostly in a local manner between themselves and with the environment. The temporal structure of a model can often be divided into several concurrent threads, where in each thread components of the system interact in "episodes" that can themselves be regarded as a discrete unit. Control propositions can then provide additional structure by classifying each episode either as "adaptive" or "performative". In this manner we can often apply game-theoretic techniques to discrete abstractions of parts of the overall model, and this analysis is often particularly useful to investigate the adaptation strategies of a system. The following example will illustrate this in more detail.

To obtain more interesting adaptation concerns we focus on a slightly more complex variant of the rescue scenario presented in Section 2. We assume that the rescue mission is performed by an ensemble E (for *ensemble* or, as we see shortly, *evaders*) consisting of a large number of robots E_i. To keep the kinematics simple we assume that each robot can instantaneously change direction, but that its speed is limited by v_{max}. We can describe its position in a cartesian coordinate system at each instant with two state variables x_i and y_i. Its velocity is given by the derivations $v_i = \dot{x}_i$ and $w_i = \dot{y}_i$ with the constraint that $\sqrt{v_i^2 + w_i^2} \leq v_{max}$.

We assume that each robot has three ingress routes into the environment on which it encounters different amounts of toxic waste. Route ρ_1 is mostly free of waste and therefore does not damage the robot; on route ρ_2 the robot incurs a 10% chance of suffering debilitating damage that causes it to fail its mission, on route ρ_3 this chance is 20%.

To obtain more interesting adaptive behaviours, we now suppose that there is a party actively opposing the rescue ensemble, either because there are raptors in the area that mistake the robot for prey, or because the rescue mission takes place in a conflict zone in which an enemy is shooting at the robot. We call this opponent H (for *hunters*) and assume that each H_j moves at constant speed v_H where supposedly $v_H > v_{max}$, but where H_j cannot instantaneously change direction but only turn with a minimum radius τ_{min}. Its cartesian coordinates ξ and χ are thus described by the following equations

$$\dot{\xi} = v_H \cos\theta \qquad \dot{\chi} = v_H \sin\theta \qquad \dot{\theta} = u \cdot v_H/\tau_{min}, \quad 0 \leq u \leq 1$$

where θ is the angle between H_j and the x-axis. It is easy to see that if the speed v_H of H_j is only slightly larger than v_{max} and if its turning radius τ_{min} is large, then it is impossible for H_j to catch any E_i. On the other hand, if v_H is much larger than v_{max} and τ_{min} is small, then a E_i that finds itself close to H_j has no hope of escaping. What might perhaps not be obvious is that even this simple hunter/evader dynamics, known under the name "homicidal chauffeur problem" generates a rich solution space that gives rise to multiple different strategies, depending on the initial conditions and parameters [20].

	ρ_1	ρ_2	ρ_3
ρ_1	0, 1	1, 0	1, 0
ρ_2	0.9, 0.1	0, 1	0.9, 0.1
ρ_3	0.8, 0.2	0.8, 0.2	0,1

Fig. 3. Payoff matrix for the choice of route

The rescue scenario now unfolds in the following way: Each hunter H_j chooses a route ρ_κ to guard and each evader E_i chooses a route ρ_μ to take. The hunters that have chosen to guard a given route then attack the evaders taking that route. If E_i can successfully evade all attacks and it is not damaged by the waste, it completes its rescue mission and receives a reward of 1, otherwise it receives a reward of 0; if a hunter H_j captures an evader it receives a reward of 1 otherwise of 0. If an evader E_i is disabled by toxic waves this results in a reward of 1 for the ensemble H that is not attributed to any H_j.

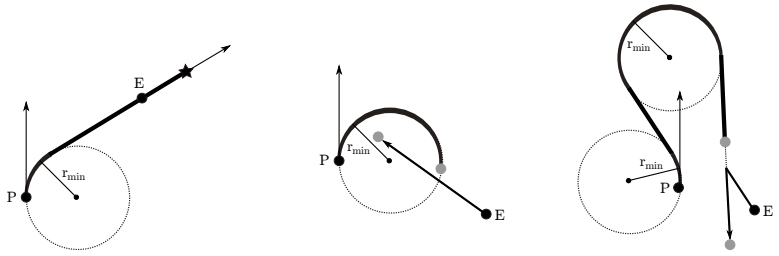
Both E and H try to optimise their respective rewards; for each trip of an evader E_i one of the players receives reward 1 and the other 0, therefore the players will always try to obtain opposite outcomes. Obviously the rewards obtained by E depend on the strategy chosen by H, and *vice versa*. Even for this relatively simple scenario, finding the optimal behaviour for the rescue robots is a non-trivial task. Therefore we have to divide the model into smaller components that are more amenable to analysis, and then further refine our initial analysis steps once we have gained a better understanding of the problem. To this end we will first focus on the choices of single agents.

When looking at the trajectory of an evader E_i through the state space, we see that it repeatedly moves through the following segments: (1) first it picks a route ρ_μ without modifying other parameters in the state space; (2) then it moves through the other dimensions of the state space without modifying its choice of route. If we choose Φ^c so that X^c contains (1) the choice of a route ρ_μ when all other parameters are kept constant and (2) the evasive manoeuvring of an evader when avoiding a hunter, we see that the trajectory for E_i repeatedly traverses the following stages: (i) a purely adaptive segment of the trace in which E_i chooses the route it takes (the "adaptation stage"); (ii) a basic segment of the trace in which E_i performs a rescue mission, possibly interrupted by E_i evading a hunter (the "performance stage"), the evasion being an adaptive segment of E_i trajectory that is not purely adaptive. This structure supports the design of the ensemble relatively well: The non-adaptive behaviours that are required in the basic segment (navigation, picking up the victim, etc.) can be implemented using established methods. The adaptation in the performance stage depends only on the local interaction of an evader and a hunter, and it can be solved for different initial configurations of E_i and H_j. If the developers of the evaders know the control strategies that are used by the hunters, the designers of the evaders can exploit this model to develop efficient strategies to counter them, or determine that no such strategies exist.

The adaptive segment is, in some respects, the most challenging part: It cannot be locally solved since it depends crucially on the distribution of hunters between the different routes, and on the expected outcomes of the "homicidal chauffeur" game between hunters and evaders. In order to restrict the amount of required game-theoretic background, we study a greatly simplified version of this question. We assume, for now, that a single hunter $H = H_1$ is matched against a single evader $E = E_1$ and that the hunter always manages to capture the evader when they choose the same route. Fig. 3 shows the payoff matrix for this game. The rows of this matrix represent the route choices of E, the columns the choices of H. In each cell of the table, the first number gives the reward obtained by E if this pair of strategies is played, the second number gives the reward of H. For example, if H and E both choose route ρ_1 then H receives a reward of 1 since we assume that it always captures E, and E receives a reward of 0. If, however, E chooses ρ_2 while H chooses ρ_1, then 90% of the time E receives a reward of 1, whereas 10% of the time it is disabled by toxic waves, and therefore H receives the reward even though the two players do not encounter each other. See [18] for more information on utility theory and values of lotteries.

Given this payoff matrix, how should E choose its routes? It is clear that no choice of route of E is a best reply against all possible choices of H, since the pair of routes ρ_i, ρ_i always results in a loss for H. The solution for robot E is therefore to mix the routes it chooses, so that H cannot predict the route that it should guard. Closer investigation of the game results in the observation that to prevent H from exploiting its choices E has to pick the probability p_i of choosing route ρ_i so that its probability of survival times the probability of choosing the route is the same for all routes [22]: $1 \times p_1 = 0.9 \times p_2 = 0.8 \times p_3$. This results in the approximate probabilities $p_1 = 30\%$, $p_2 = 33\%$ $p_3 = 37\%$. Therefore, perhaps slightly counter-intuitively, E has to choose the most dangerous route most frequently to maximise its guaranteed reward. To refine this analysis we need to compute the expected outcomes of the "homicidal chauffeur" (HC) game between E and H. Whereas the choice of route could be cast in terms of a traditional competitive game this is not possible for the HC game; here the continuous time and the dynamics generated by the system of differential equations are essential. We are therefore in the realm of differential games [15].

The HC game generates a rich and varied solution space that we only describe qualitatively and in general terms: If the hunter H arrives in a position behind and only slightly offset to the evader E as shown in Fig. 4(a), the best solution for E is to move away from H in a straight line, and H will follow this line until it captures E (indicated by the star). If the HC game continues indefinitely long, the velocity of H is much higher than the maximal velocity v_{max} of E and the turn radius τ_{min} of H is sufficiently small this capture will always occur and the strategy employed by E does not actually matter given the reward structure of our model. But if H has only a limited range (e.g., due to a small fuel supply), and the initial position of E is far enough away, E may successfully evade H. The goal of E in the HC game is therefore to maximise the distance that H has to cover before capture can no longer be avoided.



(a) Chase (b) Direct approach (c) Swerve

Fig. 4. Strategies for initial configurations: evader in back front (a) and back (b,c)

If the initial configuration is slightly different, with E behind H, a different situation may arise: E may navigate *towards* H in order to arrive inside the circle described by the turn of H, as shown in Fig. 4(b). Once E is inside the turning circle of H, the hunter cannot reach the evader by continuing to turn in the same direction. But this does not mean that E will necessarily escape H: the hunter can "swerve" away from E, i.e., turn in the opposite direction, then continue straight to gain some distance and, once it has gained enough separation, turn back towards E as shown in Fig. 4(c). In this case the correct strategy for E is to first turn towards H in order to force H into as big a swerve as possible, and then to turn away from H once the chase situation of Fig. 4(a) has been reached. By formalising these observations we can compute the probability that E can escape H for a given distribution of initial configurations. It is thus possible to use this estimate to improve the choice of ingress route by inserting these values into the payoff matrix 3.

So far the analysis has only taken into account the interaction between two players. This is not a problem for the HC game, since the pursuit dynamics of the system consists of a number of independent two-player HC games. But for the choice of route, the relative sizes of the ensembles E and H can greatly influence the validity of the analysis: If the cardinalities of H and E are similar then the analysis remains at least approximately valid for many reasonable strategies. If, however there are three times as many members in H than in E, then the system H has a strategy that prevents E from completing any rescue mission at all (guard each route with one third of H's members) and the analysis for the single-robot case cannot be transferred.

There are two additional complications when considering the adaptation of competing ensembles consisting of multiple members each: (1) When there are multiple players in each "team" E and H, there is no guarantee that a stable equilibrium of strategies will emerge; instead, at each point of time E is trying to find a best strategy against the mix of strategies currently played by the members of H while simultaneously H is trying to find a best strategy against the mix of strategies currently played by the members of E. This results in a dynamical system that may exhibit stable behaviour, periodic cycles between

different strategy mixes, or even chaotic changes in the strategy mixes of E and H. (2) The previous analysis relies on the assumption that both players play optimal strategies. This is not necessarily the case when a system is deployed in a real environment, and assuming that opponents play optimally when this is not often the case leads to less than optimal performance.

For these reasons, systems will rarely be developed with fixed adaptation rules; instead, members of an ensemble will often observe which strategies (used either by themselves or by other members) are particularly successful and use these strategies more frequently in the future. The effects of these kinds of adaptations can also be modelled and analysed in a game-theoretic setting: Since it is possible to compute the reward obtained by different strategy combinations in the basic segment of our example, we can compute the expected reward for a strategy used by a rescue robot E_i against a distribution of strategies for H. If we have large numbers of evaders and hunters, we can regard the sequence of adaptations as an evolutionary game: We abstract from the detailed interleaving of the segments of individual agents and assume that adaptation is a continuous game played between randomly chosen evaders and hunters. We assign a probability of being chosen to each of the paths ρ_μ for $E_i \in E$ and for $H_j \in H$: The expected value of a choice of ρ_μ against the distribution of H (respectively E) in the adaptation stage can be computed using the corresponding performance stages. The probabilities in the next round of the game are adjusted using these values: strategies with high values in the previous stage are played more frequently in the next stage; strategies with low values are played less frequently.

Chapter II.4 [12] in this volume goes into more details about evolutionary games and the methods to solve them. Note that the separation of behaviours into adaptive and base behaviours allowed us to reduce the analysis task into the pairwise evaluation of configurations (which depends on the number of configurations but not on the number of agents), and an evolutionary game between two populations of configurations (which, again, depends on the number of configurations but not the size of the ensembles). Since neither part of the analysis depends on the size of the ensembles, they scale to ensembles of arbitrary size. Furthermore, while the computation of the values of the performance stages is rather involved, this computation can be performed offline; for a moderate number of configurations the computation of an adaptation strategy via the evolutionary game becomes viable at run time. This is important if the evaders need to dynamically adjust their adaptation strategy (e.g., because there is uncertainty about the distribution of strategies used by the hunters).

We conclude this section by returning to the connection between black-box and white-box adaptation mentioned in Section 4.1: Suppose that we start with a simple system that supports neither the choice of route nor the evasive manoeuvring described in this section. By adding either of these features to the implementation we allow the system to move along trajectories through its control space X^C that were not present in the original system. The new trajectories in turn allow the system to operate in adaptation spaces in which it previously could not reach its goals, i.e., they increase its black-box adaptivity. Similarly,

if we want to increase the system's black-box adaptivity so that it can operate in new environments, we have to add new trajectories through its phase space. Since these trajectories serve the express purpose of adapting the system to novel situations it is sensible that the parts of the phase space traversed by these trajectories belong to the control space X^C. This shows again that black-box and white-box adaptation are closely connected when the control propositions of the system are appropriately defined.

5 Related Work

The notion of behavioural adaptation has been studied in several works. The interested reader is referred to [7,10,23]. For a discussion of work related to the main formalisms used in this paper like GEM [14] and AIAs [6] we refer to the corresponding publications. We briefly discuss here some works related to properties of adaptive systems, a key notion in the presented contribution.

Several proposals follow a black-box perspective that aims at somehow measuring or expressing requirements on how a software system changes its ability to reach a goal under specific context variations. An interesting contribution of this kind is represented by [17], which analyses the notion of adaptation in a very general sense and identifies the main concepts around adaptation drawn from several different disciplines, including evolution theory, biology, psychology, business, control theory and cybernetics. Furthermore, it provides some general guidelines on the essential features of adaptive systems in order to support their design and understanding. The author claims that *"in general, adaptation is a process about changing something, so that it would be more suitable or fit for some purpose that it would have not been otherwise"*. Accordingly, the term *adaptability* is then used to denote the capacity of enacting adaptation, and *adaptivity* for the degree or extent to which adaptation is enacted. The author concludes by suggesting that *"due to the relativity of adaptation it does not really matter whether a system is adaptive or not (they all are, in some way or another), but with respect to what it is adaptive"*.

A formal black-box definition is proposed in [5]. If a system reacts differently to the same input stream provided by the user at different times, then the system is considered to be adaptive because ordinary systems should exhibit a deterministic behaviour. Thus, a non-deterministic reaction is interpreted as an evidence of the fact that the system adapted its behaviour after an interaction with the environment. For example, a system where a change of behaviour is triggered by an interaction with the user would not be classified as adaptive, which we think is too strong a requirement.

A different line of research studies the properties of adaptive systems and their classification according to the kind of computations that are concerned with, so that the usual adaptation analysis $S \Vdash \mathcal{A}$ is instantiated in some of the computations of S depending of the class of goals in $\mathcal{A}$. We have seen this in the trace-based semantics of GEM presented here.

Some authors [29,28,16] distinguish the following three kinds of properties. *Local* properties are "*properties of one [behavioral] mode*", i.e. properties that must be satisfied by basic computations only. *Adaptation* properties are to be "*satisfied on interval states when adapting from one behavioral mode to another*", i.e. properties of adaptation phases. *Global* properties "*regard program behavior and adaptations as a whole. They should be satisfied by the adaptive program throughout its execution, regardless of the adaptations.*", i.e. properties about the overall behaviour of the system.

The authors of [6] consider also the class of *adaptability* properties, i.e. properties that may fail for local (i.e. basic) computations, and that need the adapting capability of the system to be satisfied. We refer to [6] for a presentation of some such properties in the context of AIAs.

6 Conclusion

The development of reliable autonomic computing systems poses many challenges for the software engineer. Several approaches have been proposed to tackle those challenges at different stages of system development and regarding the various aspects of those systems. In this paper we focused on behavioural adaptation aspects as tackled in requirements engineering, modelling and programming activities within development methodologies such as the one of ASCENS (cf. Fig.1).

We have reconciled two foundational approaches to behavioural adaptation, each taking a different perspective: GEM which provides a black-box perspective, useful to reason about adaptivity from the requirements point of view, and AIAs, which provide a white-box perspective, useful to reason about adaptivity from the modelling and programming point of view. A first common ground to relate both approaches is a trace-based semantics. First, GEM is instantiated on a trace inclusion-based notion of property satisfaction (i.e. adaptation in the GEM approach). Second, the AIA approach allows us to distinguish adaptations from normal behaviours within traces. A notion of coherence is then defined which can be used to identify mismatches between requirements and models in the design of an autonomic system. A second, more sophisticated approach has been presented as well, built on a game-based semantics of adaptive systems, as aspirated in [8].

An interesting line of research is to investigate quantitative notions of the hereby proposed notion of adaptation coherence, e.g. to measure the influence of adaptation mechanisms to achieve certain goals that would provide system developers with further tools to asses and analyse their designs.

References

1. Abeywickrama, D.B., Bicocchi, N., Zambonelli, F.: SOTA: towards a general model for self-adaptive systems. In: Reddy, S., Drira, K. (eds.) WETICE 2012. pp. 48–53. IEEE Computer Society (2012)
2. de Alfaro, L.: Game models for open systems. In: Dershowitz, N. (ed.) Verification: Theory and Practice. LNCS, vol. 2772, pp. 269–289. Springer (2003)

3. de Alfaro, L., Henzinger, T.A.: Interface automata. In: ESEC/SIGSOFT FSE 2001. ACM SIGSOFT Software Engineering Notes, vol. 26(5), pp. 109–120. ACM (2001)
4. Autonomic Service Component Ensembles (ASCENS), http://www.ascens-ist.eu
5. Broy, M., Leuxner, C., Sitou, W., Spanfelner, B., Winter, S.: Formalizing the notion of adaptive system behavior. In: Shin, S.Y., Ossowski, S. (eds.) SAC 2009. pp. 1029–1033. ACM (2009)
6. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: Adaptable transition systems. In: Martí-Oliet, N., Palomino, M. (eds.) WADT 2012. LNCS, vol. 7841, pp. 95–110. Springer (2012)
7. Bruni, R., Corradini, A., Gadducci, F., Lluch Lafuente, A., Vandin, A.: A conceptual framework for adaptation. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 240–254. Springer (2012)
8. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: Adaptation is a game. TinyToCS 2 (2013)
9. Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A.: Modelling and analyzing adaptive self-assembly strategies with Maude. Science of Computer Programming 99(1), 75–94 (2015)
10. Bruni, R., Corradini, A., Gadducci, F., Lluch Lafuente, A., Vandin, A.: A white box perspective on adaptation. In: Nicola, R.D., Hennicker, R. (eds.) Software, Services and Systems. LNCS, vol. 8950 (2015)
11. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: The SCEL language. ACM Transactions on Autonomous and Adaptive Systems 9(2), 7:1–7:29 (2014)
12. Hölzl, M., Gabor, T.: Reasoning and Learning for Awareness and Adaptation. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project, Lecture Notes in Computer Science, vol. 8998. Springer Verlag, Heidelberg (2015)
13. Hölzl, M., Koch, N., Puviani, M., Wirsing, M., Zambonelli, F.: The EDLC and best practises for collective adaptive systems. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project, LNCS, vol. 8998. Springer (2015)
14. Hölzl, M.M., Wirsing, M.: Towards a system model for ensembles. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 241–261. Springer (2011)
15. Isaacs, R.: Differential Games. Dover (1965)
16. Kulkarni, S.S., Biyani, K.N.: Correctness of component-based adaptation. In: Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C. (eds.) CBSE 2004. LNCS, vol. 3054, pp. 48–58. Springer (2004)
17. Lints, T.: The essentials in defining adaptation. Aerospace and Electronic Systems 27(1), 37–41 (2012)
18. Maschler, M., Solan, E., Zamir, S.: Game Theory. Cambridge University Press (2013)
19. Nicola, R.D., Latella, D., Lluch Lafuente, A., Loreti, M., Margheri, A., Massink, M., Morichetta, A., Pugliese, R., Tiezzi, F., Vandin, A.: The SCEL language: Design, implementation, verification. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project, LNCS, vol. 8998. Springer (2015)
20. Patsko, V.S., Turova, V.L.: Homicidal chauffeur game: History and modern studies. In: Breton, M., Szajowski, K. (eds.) Advances in Dynamic Games, Annals of the International Society of Dynamic Games, vol. 11, pp. 227–252. Birkhäuser (2011)

21. Pinciroli, C., Bonani, M., Mondada, F., Dorigo, M.: Adaptation and awareness in robot ensembles: Scenarios and algorithms. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project. Springer (2015)
22. Ross, D.: Game theory. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Stanford University, Winter 2014 edn. (2014), http://plato.stanford.edu/archives/win2014/entries/game-theory/
23. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Transactions on Autonomous and Adaptive Systems 4(2), 14:1–14:42 (2009)
24. Salvaneschi, G., Ghezzi, C., Pradella, M.: Context-oriented programming: A programming paradigm for autonomic systems (v2). CoRR abs/1105.0069 (2012)
25. Serbedzija, N.: The ASCENS case studies: Results and common aspects. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project, LNCS, vol. 8998. Springer (2015)
26. Vassev, E., Hinchey, M.: Requirements engineering. In: Wirsing, M., Hölzl, M., Koch, N., Mayer, P. (eds.) Software Engineering for Collective Autonomic Systems: Results of the ASCENS Project, LNCS, vol. 8998. Springer (2015)
27. Zadeh, L.A.: On the definition of adaptivity. Proceedings of the IEEE 51(3), 469–470 (1963)
28. Zhang, J., Goldsby, H., Cheng, B.H.C.: Modular verification of dynamically adaptive systems. In: Moreira, A., Schwanninger, C., Baillargeon, R., Grechanik, M. (eds.) AOSD 2009. pp. 161–172. ACM (2009)
29. Zhao, Y., Ma, D., Li, J., Li, Z.: Model checking of adaptive programs with mode-extended linear temporal logic. In: EASe 2011. pp. 40–48. IEEE Computer Society (2011)